論 文

다중프로세서 시스템에서 파이프라인 전송 버스의 설계 및 성능 평가

正會員 尹 龍 鎬* 正會員 林 寅 七*

Design of Pipeline Bus and the Performance Evaluation in Multiprocessor System

Yong Ho Yoon* In-Chil Lim* *Regular Members*

要 約

본 논문은 단일 버스를 사용한 밀결합 다중프로세서 시스템에서 데이타 전송량을 극대화하기 위해 파이프라인 전송 기능을 가지는 버스 프로토콜를 제안한다. 이 버스는 어드레스와 데이타의 전송을 위해 독립적인 전송 통로와 블록 전송기능을 두고, 최대 264 Mbytes/sec 데이타의 전송 능력을 가진다. 이 버스를 기반으로 각각의 프로세서 보드의 내부에 캐쉬를 포함한 다중프로세서 시스템을 모델링하고, 시뮬레이션를 통해 캐쉬의 메모리의 참조율 변화에 따른 버스의 성능 및 시스템의 성능을 평가한다.

본 버스를 이용할 경우 10개 까지의 프로세서 보드가 버스에 장착되어도 버스가 포화되지 않고, 4개 까지의 메모리의 인터리빙에 대하여 성능이 선형적으로 증가함을 알 수 있다.

ABSTRACT

This paper proposes the new bus protocol in the tightly coupled multiprocessor system. The bus protocol uses the pipelined data transfer and block transfer scheme to increase the bus bandwidth. The bus also has the independent transfer lines for the address and data respectively, and it can transfer the data up to maximum 264 Mbytes/sec. This paper also models the multiprocessor system where each processor boards have the private cache. Simulation evaluates the bus and system performance according to hit ratio of the reference data in cache memory. In the case of using this bus, the bus is evaluated not to be saturated when up to 10 processor boards are connected to the bus. As for up to 4 memory interleavng, the performance increases linearly.

*漢陽大學校 電子工學科
Dept. of Electronic Eng., Hanyang University
論文番號 : 93-31

I. 서 론

최근 값싸고 성능이 뛰어난 마이크로프로세서들의 출현으로 기존의 컴퓨터 시스템보다 가격 대 성능 비

가 우수하고 신뢰도가 높은 공유메모리 다중프로세서 시스템의 개발이 가속화되고 있다. 상용화된 대부분의 공유메모리 다중프로세서 시스템은 그림 1과 같이 여러개의 마이크로프로세서와 공유메모리들을 단일 버스로 연결시킨 형태이다. 그러나 이러한 형태의 시스템 구조는 다음과 같은 세가지의 문제가 있다. 첫째는 메모리의 충돌 문제로 하나의 메모리 모듈이 오직 하나의 메모리 요청만 처리할 수 있기 때문에 여러 프로세서에서 들어오는 요청들이 순서적으로 처리되어야 한다. 즉, 시간적으로 우선한 요청만이 처리되고 나머지 요청들은 재시도 되며 재시도되는 요청은 시스템 버스를 사용하지만 실제 데이타 전송은 이루어지지 않는다. 따라서 다른 요청에 유용하게 사용될 버스를 낭비하는 결과를 초래하여 전체 시스템의 성능 저하를 가져온다. 둘째, 메모리의 요청이 비록 서로 다른 모듈이더라도 하나의 상호 연결망으로 공유하므로서 생기는 통신 충돌의 문제이다. 세째는 위의 두 문제의 결과로 발생하는 메모리의 요청에 대한 응답지연이 프로세서의 수행속도를 지연시켜 결과적으로 시스템의 성능 저하를 가져온다.

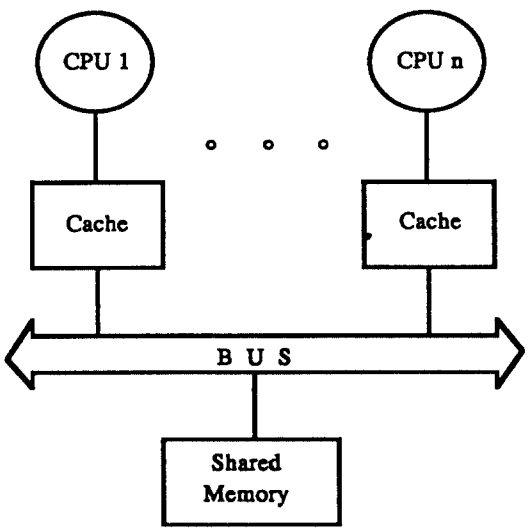


그림 1. 단일 버스를 기반으로 한 다중프로세서 시스템
Fig. 1. Multiprocessor System based on Single Bus

첫번째 문제는 공유메모리의 모듈 수를 증가시키거나 모듈 내에 뱅크 수를 늘려서 인터리빙 효과를 최대화하거나 참조 적중율이 높은 캐쉬를 각 프로세서가 갖도록 해서 해결할 수 있다. 캐쉬를 사용하는 방법은 두번째 문제를 해결하는 하나의 방법이다. 그러나 단일 프로세서 시스템에서 버스를 사용하는 현상과는 달리, 단일 버스를 기반으로 한 다중프로세서 시스템에서는 여러 프로세서들이 동시에 버스를 사용하고자할 때 버스 충돌은 불가피하게 발생한다. 동시에 버스를 요청한 프로세서들 중에서 한개의 프로세서만이 선택되어 버스를 사용하고, 나머지 프로세서들은 버스 사용이 가능할 때까지 대기하여야 한다. 버스 충돌로 인한 메모리의 참조 시간 지연은 다중프로세서 시스템의 성능에 많은 영향을 미치기 때문에 단일 버스를 기반으로 한 다중프로세서 시스템에서 버스의 데이타 전송 프로토콜은 단일 프로세서를 갖는 시스템의 버스 전송 프로토콜과는 달리 데이타 전송량을 최대화하도록 설계되어야 한다.

파이프라인 전송 프로토콜은 메모리의 참조를 위한 버스의 동작이 버스 상에 파이프라인되어 동시에 여러 메모리의 참조 요청이 버스를 통해 수행될 수 있다. 그러나 다중프로세서 시스템의 버스가 파이프라인 전송 기능을 이용하여 데이타의 전송량을 증가시킬 수 있지만, 앞에서 설명한 메모리의 참조 충돌과 통신 충돌이 발생할 수 있어 이것은 시스템의 성능을 저하시키는 또 다른 요소가 된다. 메모리의 참조 충돌과 통신 충돌의 원인은 기본적으로 각 프로세서의 메모리의 참조율이 높은데에 그 이유가 있으며, 단일 버스를 가진 다중프로세서 시스템에서는 참조율를 줄이기 위한 방법으로서 캐쉬[1]를 이용하고 있다. 캐쉬의 사용 배경에는 프로그램의 수행 특성인 지역성(locality)에 있다. 즉, 한번 사용한 데이타는 가까운 시간내에 다시 사용될 확률(temporal locality)이 높고, 또한 한번 사용된 데이타에 인접한 데이타는 곧 다시 사용될 가능성(spatial locality)이 크다. 캐쉬는 최근에 사용한 메인 메모리의 일부 데이타를 가지고 있으며 프로세서가 메모리를 참조하고자할 때 먼저 캐쉬에서 참조하고(hit), 캐쉬에 없을때는 버스를 통해 메모리에서 해당 데이타를 가져와 캐쉬에서 참조하게 된다. 즉, 캐쉬에 없는 데이타를 버스를 통해 메모리에 있는 데이타를 참조할 확률을 줄이면 버스의 트래픽(traffic)을 감소시키고, 주어진 버스에서 프로세서의 증가에 따라 시스템의 성능이 비례하여 증가하고, 메모리의 접근 시간을 감소시키므로서 프로세서의 효율를 증가시킬 수 있다. 그림 2는 캐쉬가 없을때 메모리의 참조 흐름을 보여 준 것으로 모든 메모리의 참조가 버스를 통해 이루어진

다. 그림 3은 캐쉬가 있을 경우로 캐쉬의 참조 적중 여부에 따라 버스를 사용하므로서 버스의 사용율를 줄일 수 있다.

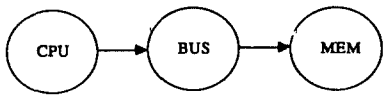


그림 2. 캐쉬가 없을 때의 메모리 참조 흐름
Fig. 2. Memory Reference Without Private Cache

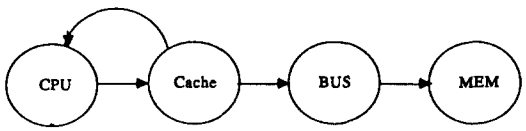


그림 3. 캐쉬가 존재할 때의 메모리 참조 흐름
Fig. 3. Memory Reference With Private Cache

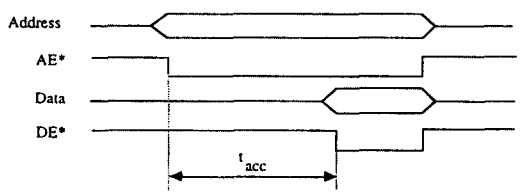
파이프라인 프로토콜를 사용하는 사용화된 시스템에서는 Symmetry[2], Balance[3][4] 등이 있다. 이들의 버스의 사양은 제공되지 않기 때문에 정확한 버스의 동작은 알 수 없으며, 다만 버스의 어드레스 라인를 데이타와 멀티플렉싱하고 있다는 사실만 알려지고 있다. 그러나 최근에 개발된 마이크로프로세서는 고성능화되고 메모리 소자의 접근 속도도 빨라져 버스의 사용량이 크게 증가하고 있는 상황에서 이러한 방식을 채택하였을 경우 버스의 대역폭(bus bandwidth)에 제한를 받게 되어 실제로 100 Mbytes /sec 이상의 대역폭을 기대하기 어렵다.

본 논문에서는 단일 버스를 기반으로 한 다중프로세서 시스템에서 데이타의 전송율를 극대화시킬 수 있는 독립적인 어드레스 버스와 데이타 버스의 라인을 갖고, 블록전송 기능을 지원하는 파이프라인 전송 프로토콜의 시스템 버스를 설계한다. 버스에 연결된 각 프로세서의 캐쉬의 참조 실패율(miss rate)에 따라 버스의 사용 효율, 데이타의 접근시간, 프로세서의 효율를 시뮬레이션을 통해 고찰한다.

본 논문은 서론에 이어 2장에서는 단일 버스의 다중프로세서 시스템에서 버스 점유형 프로토콜과 파이프라인 전송 프로토콜를 데이타의 전송 방식으로 이용하였을 경우에서의 성능을 비교한다. 3장에서는 본 논문에서 제안한 고속 파이프라인 전송 프로토콜을 제시한다. 여기에서는 버스의 중재방법, 데이타 읽기/쓰기의 방식, 메모리의 참조 요구에 대한 응답 방법, 블록 데이타 전송을 설명한다. 4장에서는 파이프라인 전송 프로토콜를 사용한 시스템 버스와 캐쉬가 내장된 프로세서들이 연결되어 있는 시스템을 모형화한 시뮬레이터에 주어질 입력부하 모형에 대해 기술한다. 5장에서는 시뮬레이션 과정과 결과에 대해 설명하고, 6장에서 결론을 기술한다.

Ⅱ. 버스 점유 방식과 파이프라인 전송 방식의 비교

버스 점유 프로토콜은 그림 4와 같이 버스 동작을 시작한 프로세서가 버스 동작을 완료할때 까지 계속적으로 버스를 점유하는 방법이다. 이때 t_{acc}로 표시된 것은 메모리가 어드레스를 받은 후 해당 데이타를 준비하는 시간이며 버스 동작은 전혀 이루어지지 않는 시간이다. 파이프라인 전송 프로토콜은 버스 점유 프로토콜에서의 t_{acc}시간을 다른 프로세서의 버스 동작에 사용 가능하도록 버스를 점유하지 않고, 데이타 요청 단계와 데이타 응답 단계가 분리된 형태이다.



t_{acc} : 메모리가 데이터를 준비하는 시간,
AE : Address Enable, DE : Data Enable

그림 4. 버스 점유형 프로토콜
Fig. 4. Bus-Holding protocol

단일 프로세서 시스템에서 일반적으로 사용되고 있는 버스의 전송 프로토콜인 버스 점유(Bus-holding)방식은 전송 프로토콜이 간단하여 구현 비용이 저렴한 장점이 있으나 다중프로세서 시스템 환경에서는 버스의 충돌시 대기해야 하는 시간이 길어지게 되어 버스의 높은 전송 대역폭을 기대하기 어렵다. 파이프라인 전송 방식은 하나의 데이타의 전송 주기를 여러개의 서브 주기로 나누어 각 서브 주기 단위로 중첩시켜 전송하므로서 버스 충돌의 대기 시간을 줄이고 데이타의 전송 대역폭을 늘릴 수 있다. 그림 5는 다중프로세서 시스템에서 두 방식에 대하여 버스에서 나타날 수 있는 전송 형태를 보여 준다. 그림에서 나타난 것처럼 단위 시간에 처리할 수 있는 데이

타의 처리량은 파이프라인 전송 방식이 매우 높음을 알 수 있다.

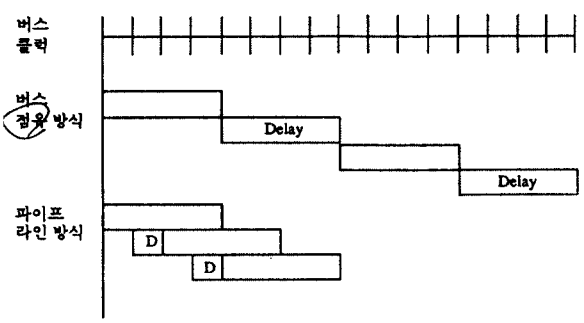


그림 5. 버스 점유 방식과 파이프라인 전송 방식의 비교
Fig. 5. Bus-Holding vs. Pipeline data transfer type

아래 식 (1)은 버스 점유 방식을 이용한 데이타 전송시간을 나타내고 있다. 즉, 전체 전송시간은 각 전송의 버스 중재 시간과 데이타 전송시간을 더한 값과 같다. 식 (2)는 파이프라인 전송 프로토콜를 이용한 데이타의 전송시간을 수식적으로 표현한 것으로서 전체 데이타의 전송시간은 각 데이타의 전송시 버스 중재시간을 더한 후 마지막 데이타의 전송시간을 더한 것과 같다.

$$T_{bus-hold} = \sum_{i=1}^{i=C_{tr}} (T_{arb} + T_{op}) \quad (1)$$

$$T_{pipeline} = (\sum_{i=1}^{i=C_{tr}} T_{arb}) + T_{op} \quad (2)$$

여기서, Tarb : 버스 중재 시간
Top : 하나의 데이타 전송 시간
Ctr : 데이타 전송 갯수

식 (1)과 (2)를 이용하여 시뮬레이션을 수행한 결과는 표 1과 같다. 시뮬레이션에서 T_{op}는 6개의 버스 클럭, C_{tr}은 128로 정하고, T_{arb}의 값을 변화시켜 표 1의 결과를 구하였다. T_{arb}는 표 1의 트래픽(traffic) 변수를 평균으로 한 기하학 분포(geometiric distribution)를 이용하여 구하였다. 트래픽은 버스의 사용 상태로서, 만일 트래픽이 0.1이면 특정 프로세서가 버스를 사용하려고 중재에 참여하였을때 성공할 확률이 90%을 의미한다. 표 1의 결과를 보면 버스의 트래픽이 증가함에 따라 성능의 차이는 더 커짐을 나타낸다. 버스 점유 방식과 파이프라인 전송 방식의 성능이 비슷하여지려면, 버스 점유 방식의 T_{op}가 파이프라인 전송 방식보다 6분의 1(B-h T_{op}) 정도로 작아야함을 알 수 있다. 따라서 다중프로세서 시스템에서 버스의 전송 프로토콜은 파이프라인 전송 방식을 이용하여야만 원하는 시스템의 성능을 얻을 수 있다.

Ⅲ. 파이프라인 전송 버스 프로토콜의 구조

본 논문에서 다루는 단일 버스를 기초로 한 밀결합 다중프로세서 시스템은 그림 6과 같이 여러 프로세서(P)와 공유메모리(M)에 의해 공유되는 버스를 통해 서로 간의 데이타 전송이 이루어지는 시스템이다[5]. 이때 서로 간의 버스 사용이 충돌하는 경우를 방지하기 위해 중재 버스를 사용하고, 데이타의 전송을 위해 데이타 전송 버스를 사용한다. 특히, 데이타와 어드레스가 분리되어 동작 가능하도록 중재 버스도 분리된 형태이다. 데이타의 전송 요청은 프로세서(P)에서 발생하여 공유메모리(M)의 내용이 전송되지만 그 사이에 RQ(ReQuester)와 RP(ResPonder)에 의해 버스 동작이 수행된다. 따라서 버스 동작을 발생하여 전송 결과를 유용하게 사용하는 것은 프로세서이지만 버스 동작을 수행하는 것은 RQ와 RP이다[6].

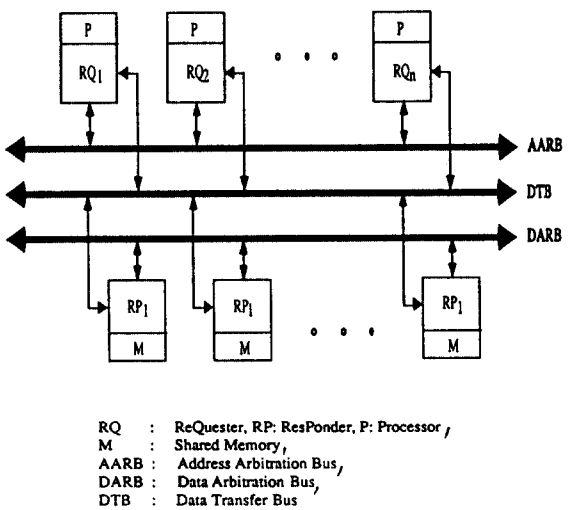


그림 6. 밀결합 다중프로세서 시스템의 모형
Fig. 6. A Model of Tightly coupled Multiprocessor System

3.1 중재

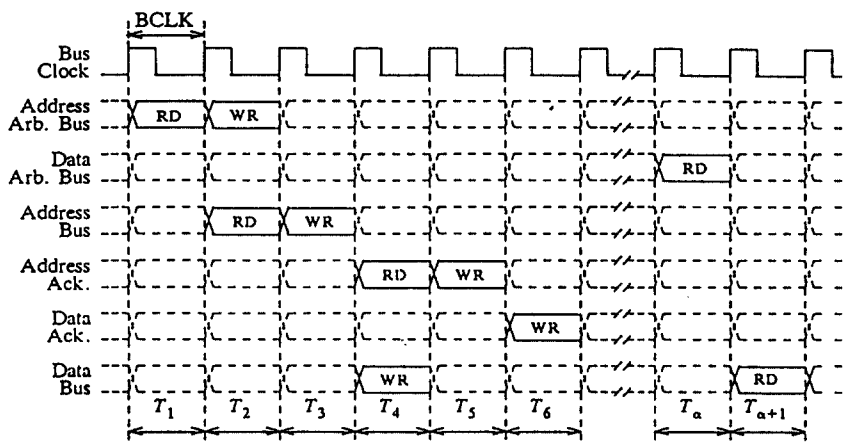
여러 프로세서에서 발생하는 메모리의 참조 요청들은 각각의 RQ에 의해 버스 동작으로 수행되며 이때 두개 이상의 버스 동작이 충돌되지 않도록 보장해야 한다. 본 버스에서는 어드레스 버스와 데이타 버스를 독립적으로 사용하고 파이프라인 전송을 함에 따라 각각의 중재가 필요하다. 모든 버스의 동작에 앞서 중재를 수행하며 각각의 중재 동작은 하나의 버스 클럭안에 수행된다. 중재 결과로 버스의 사용권을 획득한 경우 계속되는 다음 버스 클럭부터 데이타의 전송을 시작한다. 중재는 하드웨어적으로 중재가 필요한 보드의 수 만큼 독립적인 신호가 있고, 각 보드는 이 신호선 중 하나를 할당받아 버스 사용이 필요할 때 해당 신호선을 구동함으로서 중재가 시작된다. 동시에 구동된 여러 신호중 하나를 선택하는 방식은 고정 우선순위에 따라 결정된다. 버스 중재에 실패하였을 경우와 다른 프로세서가 메모리를 사용함으로 인해 메모리의 참조가 실패하였을 경우 프로세서 보드는 버스 중재를 재시도해야 한다. 그러나 메모리의 처리 시간이 재시도 주기보다 느릴 경우와 같은 메모리에 여러 RQ들이 참조하고자 할때는 바로 연속된 재시도는 버스의 효율만 나쁘게 한다. 또한 재시도 주기가 다른 요청기의 요청 주기의 공배수인 경우는 서로 주기가 맞물리는 영원한 재시도 현상이 발생할 수 있다. 이러한 현상을 막기 위해 재조정 주기를 임의로 바꾸는 것이 필요하다. 본 버스에서는 난수 발생을 사용하여 변화 시켜 바로 재시도하거나, 다음 주기에 재시도하도록 한다.

3.2 데이타 전송 방식

본 논문에서 사용한 데이타 전송 방법은 그림 7에서와 같다. 이것은 독립적인 읽기 전송과 쓰기 전송이 버스에 중첩되어 수행된 경우이고, 특히 점선으로 표시된 부분들은 다른 버스의 동작에 사용될 수 있는 상태이다[4]. 즉, 파이프라인 전송 프로토콜은 여러 버스의 동작이 중첩되어 수행될 수 있는 장점을 갖고 있어 버스 점유형 보다 높은 전송 속도를 제공할 수 있는 특징이 있다.

3.2.1 읽기 전송

읽기 전송은 그림 7에서 RD로 표시된 동작들로 수행된다. 프로세서로 부터 읽기 전송의 요청이 있게 되면, RQ는 우선 어드레스 버스의 중재 동작(T_1)을 수행하며, 이때 모든 중재는 버스의 클럭 단위로 이루어진다. 중재 주기에 이어서 RQ는 어드레스 주기를 수행한다. 어드레스 버스의 사용권을 획득한 중재 주기에 이어지는 다음 버스 사이클(bus cycle) T_2에 어드레스와 그것과 관련된 정보(읽기/쓰기 구분, 프



RD : 읽기
WR : 쓰기 전송

그림 7. 파이프라인 프로토콜의 타이밍
Fig. 7. Timing of Pipeline Protocol

로세서 식별자, 기타)를 버스에 구동한다. 어드레스 구동에 이어지는 버스 사이클 T_3은 쓰기 동작때 데이타 구동에 할당되는 것이므로 읽기 전송때는 신호 구동을 않는다. RQ가 어드레스를 구동한 버스 사이클 T_2의 끝 부근에서 버스상의 각 RP들은 버스상의 어드레스를 일시 저장하고 자신과 관련된 것인지를 계속되는 버스 사이클 T_3 동안 확인하고, 선택된 RP는 이어지는 버스 사이클 T_4에 RQ로 부터 전달받은 어드레스와 그것과 관련된 정보에 대한 응답(어드레스 응답)을 구동한다. T_4의 끝부분에서 어드레스를 구동한 RQ는 어드레스 응답을 저장하여 수행된 어드레스 주기가 잘 진행되었는지 여부를 판단하고, 이때 잘 수행된 경우는 RP가 데이타를 구동할 때 까지 기다리고, 그렇치 못하면 다시 어드레스 중재 주기와 어드래스 주기를 재시도한다. RQ가 구동한 어드레스에 의해 선택된 RP는 메모리로 부터 해당 어드레스의 내용을 준비하고, 데이타 중재 동작 (T_a)을 수행하며, 중재 주기에 이어서 RP는 데이타 주기를 수행한다. 데이타 버스의 사용권을 획득한 중재 주기에 이어지는 다음 버스 사이클 T_{a+1}에 데이타와 어드레스 주기에 저장해 놓은 프로세서 구별자를 버스에 구동한다. 어드레스 주기를 완료하고 데이타를 기다리는 RQ는 각 버스 사이클의 끝 부분에서 데이타와 그 관련 버스 정보를 저장하고, 자신의 프로세서 구별자가 나타날때 전달된 데이타를 프로세서로 돌려줌으로서 읽기 전송을 완료한다.

3.2.2 쓰기 전송

쓰기 전송은 그림 7에서 WR로 표시된 동작들로 수행된다. 버스 동작의 순서는 읽기 전송과 거의 동일하며, 어드레스 주기에 RQ가 데이타로 구동하는 것과(어드레스 주기와 구별하기 위해 "어드레스-데이타 주기"라고 함), 선택된 RP는 데이타를 받아서 자신의 메모리에 쓰기를 하는 것이 읽기 전송과 다르다. 프로세서로 부터 쓰기 전송 요청이 있으면, RQ는 우선 어드레스 버스의 중재 동작(T_2)을 수행하며, 중재 주기에 이어서 RQ는 어드레스-데이타 주기를 수행한다. 어드레스 버스의 사용권을 획득한 중재 주기에 이어지는 T_4 버스 사이클에 데이타를 버스에 구동한다. 읽기 동작때와 같은 방법으로 선택된 RP가 어드레스 응답을 T_5 버스 사이클에서 버스에 구동하고 이어지는 T_6 버스 사이클에는 전달 받은 데이타 정보에 대한 데이타 응답을 버스에 구동한다. RQ는 어드레스 응답과 데이타 응답이 모두 잘된 것으로 확인한 후 쓰기 동작을 완료하고, 그렇치 못한 경우는 처음부터 다시 쓰기 동작을 재시도한다.

RQ가 수행하는 쓰기 전송은 데이타 버스의 중재 없이 데이타 버스를 사용하고, RQ의 읽기 전송에 의해 RP에서 수행해야 될 데이타 주기는 데이타 중재 주기를 수행한 후 데이타 버스를 사용하므로 이들 둘은 충돌할 가능성이 있다. 이러한 경우를 방지하기 위해 RP는 데이타 버스의 중재에서 데이타 버스의 사용권을 획득한 경우라도 RQ의 쓰기 동작에 의한 데이타 구동과 충돌할 가능성이 있는 경우는 데이타 구동을 포기하고 데이타 중재 주기부터 다시 수행하도록 한다.

3.2.3 응답 방법

파이프라인 전송 프로토콜은 어드레스 구동과 데이타 구동이 분리된 형태로 버스 상에 여러 RQ와 여러 RP가 존재하며 진행된 버스 동작이 잘 된 것인지 여부를 판단할 필요가 생긴다. 어드레스 중재 후 어드레스를 구동한 RQ가 해당 어드레스가 특정 RP에 전달된 것을 확인하는 방법으로, 어드레스에 의해 선택된 RP가 어드레스 응답을 하도록 규정한다.

- No-error : 전달된 어드레스에 이상이 없고, 어드레스에 의해 선택된 RP는 해당 버스 동작을 수행함.
- Error : 전달된 어드레스에 이상이 있음.
- Busy : 어드레스에 의해 선택된 RP가 다른 버스 동작에 의해 사용 중임.

RQ가 쓰기 전송을 할때 데이타를 구동하며 이때 어드레스에 의해 선택된 RP는 전달 받은 데이타의 상태를 RQ로 알려준다.

- No-error : 전달된 데이타에 이상이 없음.
- Error : 전달된 데이타에 이상이 있음.

3.2.4 블록 데이타 전송

일반적으로 버스의 데이타 전송폭을 캐쉬의 라인 크기와 같도록 설계하고 있으나 캐쉬의 라인 크기가 전송폭 보다 더 크게 설계해야 할 경우가 있다. 파이프라인 전송 프로토콜은 기본적으로 버스 점유 방식을 사용하지 않고, 여러번의 버스 동작을 통해 캐쉬 라인의 읽기, 쓰기 동작을 해야 한다. 각각의 독립적인 버스 동작을 할 경우 버스 트래픽에 따라 데이타의 전송시간이 길어질 가능성이 있다. 이로 인해 캐쉬에서의 버스 적중 실패값(cache miss penalty)가 증가하게 되어 프로세서의 데이타 처리 시간이 길어

지고 성능이 저하된다. 성능 저하를 막기 위해 일정 시간(4번의 데이타 전송 시간) 버스를 점유할 수 있는 기능을 제공하고, 각각의 동작이 가능한 한 최대로 중첩이 되는 프로토콜을 제안한다.(그림 8)

블록 데이타 읽기 전송은 중재 주기와 어드레스 주기까지는 단일 전송 방식과 같으나 데이타 전송 주기에 연속적으로 4개의 데이타를 전송한다. 이때 데이타 버스를 다른 RQ, RP가 사용하지 못하도록 금지(inhibit signal) 신호를 구동한다. 이때도 데이타 버스의 충돌을 막기 위해 금지 신호를 구동한다.

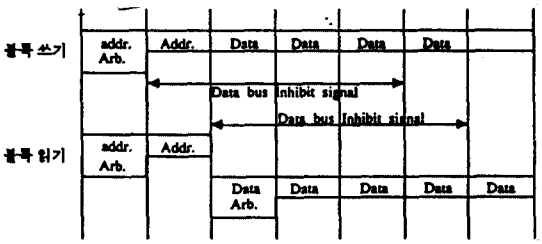


그림 8. 블록 데이터 전송 프로토콜
Fig. 8. Block Data Transfer Protocol

Ⅳ. 입력부하 모형

캐쉬가 시스템 버스에 미치는 영향은 실패율과 밀접한 관계를 가진다. 즉, 프로세서가 메모리를 참조할때 캐쉬에서 참조가 실패할 경우에만 시스템 버스를 사용하게 된다. 일반적으로 캐쉬의 특성을 알기 위한 방법은 두가지가 있다. 첫째, 프로세서와 캐쉬 그리고 버스의 모형을 만든후, 직접 실제 시스템에서 특정 프로그램을 수행하고, 이때 프로세서가 참조하는 어드레스를 추적하여 모형에 입력시켜 결과를 구하는 방법이 있고, 둘째로는 수학적 모델을 이용하여 입력부하를 구하여 모형에 적용하는 방법이 있다. 첫째 방법은 입력부하가 실제 시스템에서 수행되는 결과를 추적한 결과이므로 정확성이 매우 높은 반면 실제로 추적 어드레스를 구하기 위해서는 대상 시스템의 분석, 추적 도구의 개발, 추적 데이타의 재 정리등 어려움이 매우 많다. 두번째 방식은 어떤 특정 프로그램에 대한 것이라기 보다는 일반적인 프로그램들의 수행 특성에 대해 수학적인 모델을 만들어 입력부하를 만들기 때문에 첫째 방식에 비해 정확성은 떨어지지만 입력부하의 생성이 쉽고, 전체적인 시스템의 특성을 구하기에는 보다 용이하다. 본 시뮬레이션에서는 캐쉬의 영향을 고려하였을 때의 버스의 특성, 데이타의 참조시간 변화등의 비율를 전체적인 면에서 구하고자 하기 때문에 두번째 방식을 이용하여 입력부하를 생성하였다.

이와 관련하여 Thiebaut[7][8]의 모형은 단지 두가지의 변수만을 이용하고 있어서 간단하고, 또한 무한히 큰 용량의 캐쉬를 가정하고 있다. 즉, 캐쉬의 용량 제한으로 인한 참조 실패와 다중 캐쉬에서의 일치성(Cache coherence)으로 인한 참조 실패는 고려하지 않고 프로그램이 수행되면서 변하는 수행영역(working set)의 변화에 대한 참조 실패만을 고려하였다.

표 2. 참조 지역성에 대한 시뮬레이션 시간
Table 2. Simulation time for locality

$\theta=1.8$			$\theta=1.5$			$\theta=1.3$		
MR	A	ref.	MR	A	ref.	MR	A	ref.
0.01	1.3	57066	0.01	0.4	64000	0.01	0.125	56660
0.02	2.6	57066	0.02	0.8	64000	0.02	0.250	56660
0.03	3.9	57066	0.03	1.2	64000	0.03	0.375	56660
0.04	5.2	57066	0.04	1.6	64000	0.04	0.500	56660
0.05	6.5	57066	0.05	2.0	64000	0.05	0.625	56660
0.06	7.8	57066	0.06	2.4	64000	0.06	0.750	56660
0.07	9.1	57066	0.07	2.8	64000	0.07	0.875	56660
0.08	10.4	57066	0.08	3.2	64000	0.08	1.000	56660
0.09	11.7	57066	0.09	3.6	64000	0.09	1.125	56660
0.10	13.0	57066	0.10	4.0	64000	0.10	1.250	56660

MR : Miss Ratio ref. : Memory reference count during simulation

본 논문에서는 다중 캐쉬에서의 일치성에 의한 버스 동작이나, 제한된 캐쉬의 크기에서 발생하는 버스 동작과 같은 구체적인 캐쉬의 효과가 관심이 아니라, 정해진 캐쉬의 참조율에 따른 버스에서의 성능에 대해 고찰하기 때문에 이 모형은 훌륭한 선택 대상이다. Thiebaut의 모델에서 하나의 프로세서가 r번째 메모리 참조까지의 참조 실패의 갯수, 즉 M(r)은 식(3)으로 주어 진다.

$$M(r) = A \times r^{\frac{1}{\theta}} \qquad (3)$$

여기서 θ는 fractal dimension으로서 프로세서의 메모리 참조 지역성(reference locality)을 나타낸다. θ가 1에 가까와질수록 캐쉬에서의 메모리의 참조 실패율이 높아지고, 2에 가까와지면 지역성이 매우 높아 참조 실패율이 낮아진다. 상수 A는 작은 수로서 시간에 따른 참조 실패율를 균등하게 변화시킨다. 즉, A의 값이 크면 전체적인 실패율이 커지고, 작으면 낮아진다. 본 시뮬레이션에서는 A 값을 변화시켜서 주어진 참조 실패율에 대한 시뮬레이션 시간을 조정하였다(표2).

V. 시뮬레이션 및 결과 분석

본 시뮬레이션에서는 3장에서 제시한 파이프라인 데이타의 전송방식을 채택한 시스템 버스와 4장에서 구한 입력부하를 생성하는 프로세서와 캐쉬 시스템을 시뮬레이션 모델를 만들어 수행하였다. 사용한 시뮬레이션 언어는 C 언어로 구현된 smpl[9]을 이용하였다. 시뮬레이션 과정은 먼저 버스에 10개의 프로세서 보드가 있는 모델을 구현하였고, θ의 값에 따라 각 프로세서 별로 식 1에 의해 일련의 캐쉬에서의 메모리의 참조 성공, 실패 여부를 결정하여 입력 부하로 사용한다. 시뮬레이션 시작과 동시에 각 프로세서 별로 구한 입력부하에 따라 캐쉬에서 메모리를 참조하거나, 버스를 통해 메모리를 참조하게 된다. 버스를 통한 데이타 전송 프로토콜은 3장과 같으며, 시뮬레이션에 주어진 변수로서 프로세서가 요구한 메모리의 참조를 메모리 보드가 처리하는 시간을 읽기 요구의 경우 4 버스 클럭, 쓰기 요구의 경우 3 버스 클럭이 걸리는 것으로 정하였다.

시뮬레이션을 수행한 후의 결과로는 각각의 θ와 메모리의 참조 성공율에 대하여 버스의 사용율(Bus Utilization), 버스를 통한 메모리의 참조 실패율(Memory Busy Rate), 프로세서의 사용율(Processor Utilization), 버스를 통한 평균 메모리 참조 시간(Average Memory Access time)이다. 버스 효율를 통해 버스의 포화를 발생하는 프로세서의 갯수를 알 수 있으며, 버스를 통한 메모리 참조 실패율은 유용하지 않은 메모리 참조율를 보여준다. 프로세서의 효율은 메모리의 참조 실패로 인한 프로세서의 처리능력 낭비를 나타내고, 실패의 대가(miss penalty)를 보여준다.

본 논문에서 수행한 시뮬레이션의 절차는 다음과 같다.

단계 1: 프로세서의 갯수를 10개로, 메모리 보드가 동시에 처리할 수 있는 프로세서의 메모리 참조 수를 1개로 고정시킨 후, 각 θ 별로 캐쉬의 참조 실패율를 1%에서 10%까지 변화시켜 수행하였다.

단계 2: θ는 1.5, 캐쉬의 참조 실패율를 5%로 고정시킨 후, 프로세서의 수를 1개에서 10개까지 변화시키며 결과를 추적하였다.

단계 3: θ는 1.5, 캐쉬의 참조 실패율은 5%, 프로세서의 수를 10개로 고정시킨 후, 동시에 메모리의 참조를 수행할 수 있는 메모리의 수를 1개에서 10개까지 증가시키며 수행하였다.

단계 1은 메모리 보드가 1개인 상태에서 각 프로세서의 캐쉬에서의 참조 실패율이 변할 때의 특성을 보여주며, 단계 2는 주어진 조건 즉, 캐쉬의 참조 실패율, 목표로 하는 시스템 성능 등을 제한하였을때 버스에 연결될 수 있는 프로세서 보드의 수를 알 수 있다. 단계 3은 메모리의 인터리빙을 제공하여, 메모리에서의 충돌을 줄였을 때의 결과를 보여준다. 본 시뮬레이션에서는 하나의 프로세서 보드가 캐쉬의 참조 실패로 버스를 통해 메모리를 참조하고자 할때 대상이 되는 메모리 보드의 분포는 모든 보드에 대해 균등(uniform)하고, 캐쉬의 라인 크기 단위로 인터리빙되어 있다고 가정한다.

그림 9에서 12까지는 단계 1의 결과를 어드레스 버스의 사용율, 프로세서의 사용율, 평균 메모리의 참조시간, 메모리의 충돌율를 나타내고 있다. 각각의 θ에 대해 결과는 큰 차이를 보이지 않고 있는데, 이는 캐쉬의 효과에 의한 시스템의 성능이 캐쉬의 참조 실패율이 같을 경우 메모리의 참조 지역성과는 큰 차이가 없음을 나타낸다. 즉 참조 지역성은 캐쉬의 실패율에는 영향을 주지만, 단일 버스를 기반으로 한 다

중프로세서 시스템의 성능은 캐쉬의 참조 실패율에 따라 결정된다. 그림 9는 어드레스 버스의 사용율를 나타낸 것으로서 값이 100%에 가까와지면 버스가 포화된 것을 의미한다. 캐쉬의 실패율이 7% 이상이면 버스의 사용율이 90% 이상이 되며 10%가 되면 버스가 포화되는 것으로 나타난다. 버스 사용율이 큰 반면, 그림 11에서 보는 바와 같이 메모리의 참조 응답 시간이 매우 긴 것을 알 수 있다. 이러한 이유는 실제 버스에 나타나는 메모리의 참조 동작의 많은 부분이 전송 실패 동작이라는 것을 알 수 있다. 이 전송 실패의 분포는 그림 12에서 나타나는데, 이 버스 응답 (Busy Acknowledge) 실패는 메모리에서의 메모리의 참조 충돌에서 기인한다.

본 시뮬레이션에서는 캐쉬에서 메모리의 참조가 적중했을때 2개의 버스 사이클에 해당하는 시간이 사용된다고 가정하였기 때문에 프로세서의 사용율이 최대 50%를 넘지 못하는 것으로 결과가 나왔다(그림 10). 만일 캐쉬에서의 적중 시간을 하나의 버스 클럭이 사용되도록 설계하였을 경우 본 시뮬레이션 결과의 2배에 해당하는 프로세서 사용율를 얻을 수 있다. 단계 2의 결과인 그림 13은 프로세서의 증가에 따른 버스의 성능 변화를 보여 준다. 프로세서의 수가 5개를 기준으로 하여 평균 메모리의 처리 시간이 급격히 증가하고 있다. 버스의 사용율과 메모리의 응답 실패율이 비례하여 증가하는 것을 보면 메모리에서의 병목 현상이 심하고, 실제 버스에서 나타나는 대부분의 메모리 요청들이 실패하여 다시 재시도되는 것을 알 수 있다. 메모리의 참조 충돌율를 줄이고, 메모리에서의 병목 현상을 줄이기 위한 방법으로서 메모리의 인터리빙 방법이 있는데, 이는 데이타의 참조 처리를 병렬적으로 처리해 주고 메모리의 참조 요청을 인터리빙된 각 보드에 분산시켜줌으로서 메모리의 참조 충돌율를 줄일 수 있다. 단계 3의 결과인 그림 14의 메모리의 인터리빙 효과를 보면 6개까지의 메모리 보드의 인터리빙까지는 점진적인 성능 향상을 볼 수 있으나, 그 이상에서는 별 차이가 없는 것을 발견할 수 있다. 이 이유는 시스템의 성능이 메모리 보드에서의 충돌에 의해 성능 저하보다는 동시에 버스를 사용하고자 하는 버스에서의 충돌에 의해 좌우되기 때문이다.

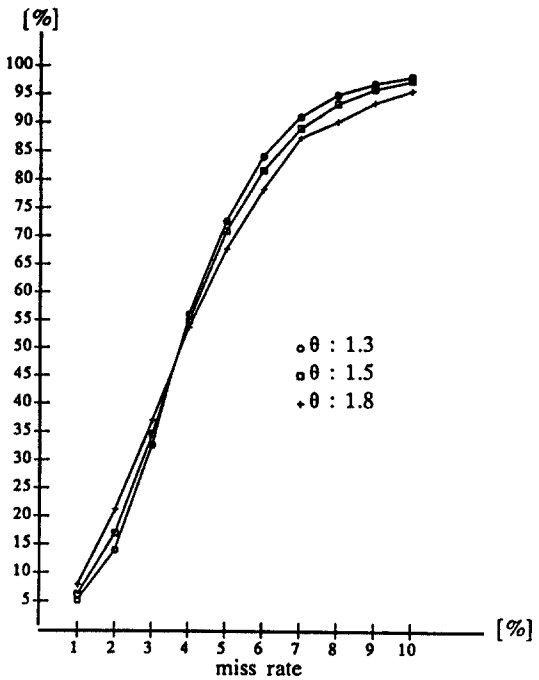


그림 9. 버스 사용율
Fig. 9. Bus Utilization

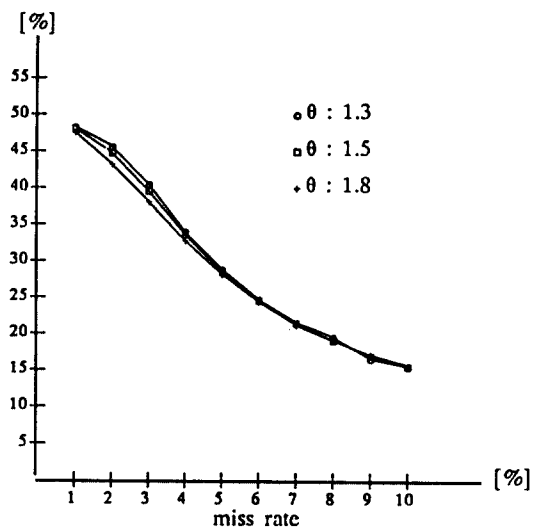


그림 10. 프로세서 사용율
Fig. 10. Processor Utilization

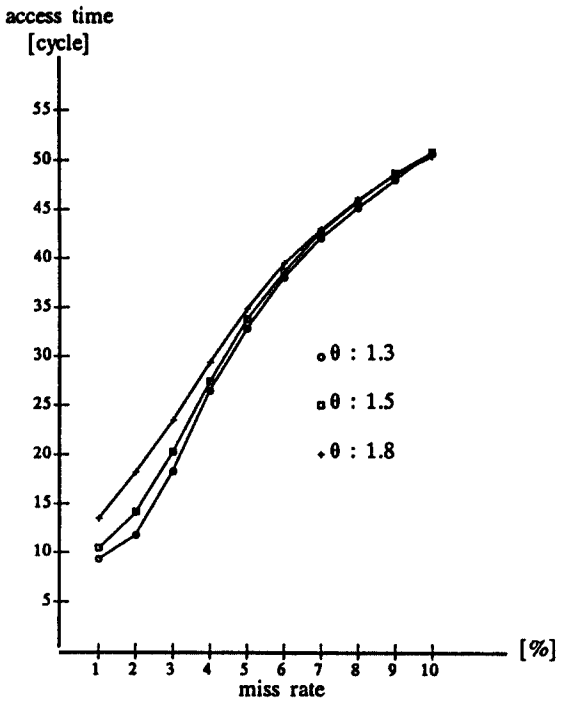


그림 11. 평균 데이타 참조 시간
Fig. 11. Average Memory Access Time

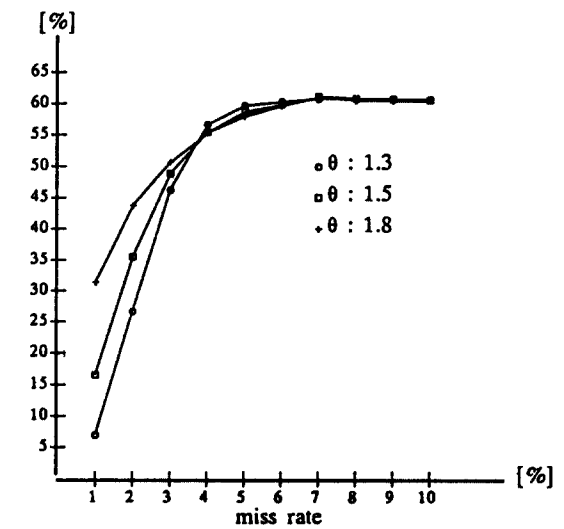


그림 12. 메모리 참조 충돌율
Fig. 12. Memory Busy Rates

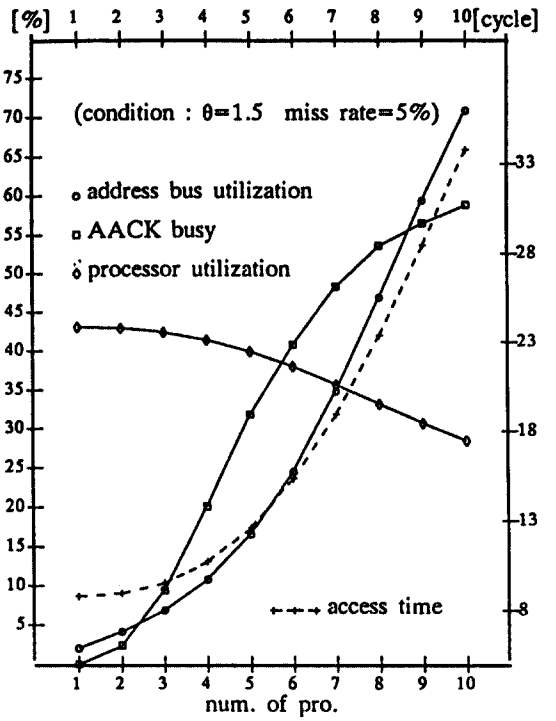


그림 13. 프로세서 수의 증가에 대한 비교
Fig. 13. Results for Processors

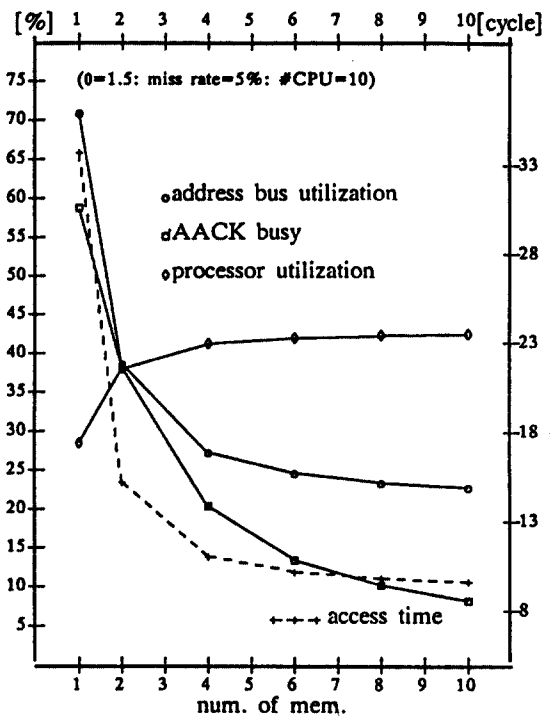


그림 14. 메모리 인터리빙의 효과
Fig. 14. Effects of Memory Interleaving

Ⅵ. 결 론

본 논문에서는 단일 버스와 공유메모리를 갖는 다중프로세서 시스템에 적용될 블록 전송 기능을 지원하는 파이프라인 전송 프로토콜의 버스 사양을 제안하였다. 이 버스를 기반으로 하여 각 프로세서 보드의 내부에 캐쉬를 가지고 있는 다중프로세서 시스템의 시뮬레이션 모델을 구현하였고, 시뮬레이션를 통해 캐쉬의 참조 실패율이 본 버스에 미치는 영향을 분석하였으며, 시스템의 성능 향상을 위한 방향을 제시하였다.

파이프라인 전송 방식은 다중프로세서 시스템의 데이타의 전송 대역폭(bandwidth)을 늘리거나, 데이타의 전송 효율(throughput)를 높이는 효과적인 방법이다. 그러나, 동일한 시간에 버스를 사용하고자 하는 프로세서가 많거나, 버스를 사용하고자 하는 빈도가 높을 때는 버스의 사용 충돌과 메모리에서의 충돌에 의해 성능의 저하를 가져오게 되므로 적중율이 높은 캐쉬를 이용하여 문제점을 해결할 수 있다. 그러나 본 시뮬레이션에서 보는 바와 같이 아무리 시스템 버스를 잘 설계하더라도 캐쉬에서의 데이타의 참조 실패율에 따라 시스템의 성능에 미치는 영향이 큼을 알 수 있다. 파이프라인 전송 방식의 버스를 기반으로 한 다중프로세서 시스템을 설계할 때는 캐쉬의 참조 성공율를 높이고, 메모리 보드의 데이타의 요구 처리시간을 줄이기 위한 방법들을 고려해야 한다.

본 논문에서 제안한 파이프라인 전송 방식의 버스를 기반으로 다중 프로세서 시스템을 구현할 경우 하나의 메모리 보드에 대하여 각 프로세서 보드 내의 캐쉬의 참조 실패율를 10% 이내로 설계하면 10개 정도의 프로세서 보드를 장착하여도 버스는 포화되지 않는다.[그림 13] 그러나 본 버스를 사용하여 시스템을 설계할 경우 시스템의 성능 조건에 따라 캐쉬의 참조 실패율에 제한을 주어 설계하여야 한다. 예를 들어 평균 메모리의 참조 시간이 버스 충돌이 없을 경우에 비해 2배 미만의 범위안에 원하는 성능을 얻을 수 있는 시스템의 설계에는 캐쉬의 참조 실패율이 3%(버스 사용율 30%) 이하가 되도록 설계하여야 한다.[그림 9] 메모리의 참조 처리 응답시간의 경우, 메모리 보드의 처리 시간을 향상하여 메모리 보드에서의 충돌을 줄이는 방법으로서 메모리의 인터리빙 방법을 도입하면 4개 까지의 메모리의 인터리빙에 대해 성능의 큰 증가를 볼 수 있으나, 그 이상에 대해서는 별 차이를 보이지 않고 있다.[그림 14] 이 이유는 시스템의 병목 현상이 메모리에서의 충돌이 아니라 버스에서의 통신 충돌로 옮겨져 시스템의 성능이 버스의 성능에 좌우되기 때문이다.

본 버스의 프로토콜를 이용하고 버스 클럭을 16.5 Mhz(60nsec bus slot)와 데이타 버스 폭을 16 bytes 설계하면 264 Mbytes/sec의 전송 대역폭을 얻을 수 있다.

참 고 문 헌

1. Alan Jay Smith, "Cache memories," ACM computing surveys, Vol.14, No.3, pp.473-530, Sep. 1982.
2. Tom Manual, "How sequent's new model outruns most mainframes," Electronics, pp.76-79, May 28, 1987.
3. Shreekant Thakkar and et. al., "Balance ; A shared memory multiprocessor system," Proc. 2nd International Conference on Supercomputing, Vol.1, pp.93-101, 1987.
4. Shreekant Thakkar and et. al., "The balance multiprocessor systems," IEEE MICRO, pp. 57-69, Feb. 1988.
5. An Do KI, Yongho Yoon, et. al., "Highly pipelined bus," proceedings JTC-CSCC '91, pp. 75-86, 1991.
6. 한국전자통신연구소, "TICOM 시스템 버스 사용자 지침서," ETRI, 1990.
7. D. Thiebaut, "On fractal dimension of computer programs and its application to the prediction of cache miss ratio," IEEE Trans. on Computers, Vol.38, No.7, pp.1012-1026, July 1989.
8. Anastasios A. Economides, et. al., "Transient Models of Bus-Based Multiprocessors," Proceedings '90 International Conference on Parallel Processing, pp.153-160, 1990.
9. M.H. MacDougall, "Simulating Computer Systems Techniques and Tools," The MIT Press, 1987.

尹 龍 鎬(Yong Ho Yoon) **正會員**
1949年 10月 15日生
1975年 2月 : 한양대학교 전자공학과 졸업(공학사)
1981年 2月 : 한양대학교 전자계산학과 대학원 졸업(공학석사)
1988年 9月 : 한양대학교 전자공학과 박사과정 수료.
1975年~1978年 : 한국과학기술연구원 연구원
1978年~1993年 現在 : 한국전자통신연구소 책임연구원
※주요관심분야 : 컴퓨터구조, 병렬처리, ASIC 설계, CAD 등.

林 寅 七(In-Chil Lim) **正會員**
第 17 卷 第 4 號 參照(통신공학회)
현재 : 한양대학교 전자공학과 교수